On Graph Classification Networks, Datasets and Baselines

Enxhell Luzhnica [*1] **Ben Day** [*1] **Pietro Lio** [1]

Abstract

Graph classification receives a great deal of attention from the non-Euclidean machine learning community. Recent advances in graph coarsening have enabled the training of deeper networks and produced new state-of-the-art results in many benchmark tasks. We examine how these architectures train and find that performance is highly-sensitive to initialisation and depends strongly on jumping-knowledge structures. We then show that, despite the great complexity of these models, competitive performance is achieved by the simplest of models – structure-blind MLP, single-layer GCN and fixed-weight GCN – and propose these be included as baselines in future.

1. Introduction

Deep learning has produced remarkable results across the full breadth of machine learning research. For the most part this has been achieved through the reapplication of the two main architectures, the CNN and RNN, adapted to two Euclidean cases – omnidirectional (image-like) and unidirectional (series) – respectively. As such there is great interest in extending the general techniques to non-Euclidean cases and graph-structured data problems in particular.

These efforts are mostly inspired by the CNN and attempting to find suitable analogs to its core components, the convolutional and pooling operators. Early work set out to develop convolution-like graph operators. The focus has now turned to developing pooling operations, often referred to as coarsening in the context of graphs. Besides static methods (Luzhnica et al., 2019), differentiable pooling frameworks have been developed. DiffPool achieved state-of-the-art (SoTA) performance across many benchmark tasks (Ying et al., 2018), however a dense representation, quadratic in memory, is required. The Graph U-Net introduces a sparse method based on pruning nodes (top-k) (Gao & Ji, 2019). Cangea et al. (2018) apply the method in graph classification by incorporating top-k pools in a GCN model, achieving performance competitive with the SoTA with scalable memory requirements.

In this work we show that, under standard initialisation (Glorot & Bengio, 2010; He et al., 2015), using the GCN and top-k operator together results in vanishing gradients beyond the first layers. In addition, we show that it is possible to attain good performance on smaller benchmark tasks simply using a global-pool[1] followed by an MLP. Furthermore, to achieve results on a par with Graph U-Net in *all* benchmarks a single-layer GCN with a jumping-knowledge (JK) connection (Xu et al., 2018) from the input graph followed by an MLP is sufficient, whether the weights of the GCN are trained or not.

Considering the implications of these results, we primarily argue for the importance of including strong, simple baselines in evaluation. We also define an initialisation scheme that remedies the vanishing gradient issue by design though we find that this does not consistently improve performance.

Motivation This work was motivated by studies of network activations and gradient flow in deeper GNNs with JK structures and top-k pooling. We found that, at initialisation, activations into the network rapidly vanish and that throughout training the gradients flowed mostly into earlier layers. These findings prompt two questions: firstly, are deeper networks only trainable thanks to JK structures bypassing later layers? and secondly, how important are the later layers to performance anyway?

2. Preliminaries

We use the standard notation: a graph $\mathcal{G}$ of N nodes with F features per node is represented by the pair $(\mathbf{A}, \mathbf{X})$ with adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$, and node feature matrix, $\mathbf{X} \in \mathbb{R}^{N \times F}$.

Graph Convolution ReLU activations and the improved GCN (Gao & Ji, 2019) are used throughout. This differs from the standard GCN in that $\hat{\mathbf{A}} = \mathbf{A} + 2\mathbf{I}$ is used i.e. self-loops have a weight of 2.

[1] A simple mean or sum over the features of all nodes.

[*]Equal contribution [1]Department of Computer Science & Technology, University of Cambridge, Cambridge, United Kingdom. Correspondence to: Ben Day <ben.day@cl.cam.ac.uk>.

Accepted at the ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data

Pooling top-k pooling is used (Gao & Ji, 2019). The pooling operator drops $N - \lceil kN \rceil$ nodes, where $k \in [0, 1)$ is a fixed hyperparameter. In all experiments this was set to 0.8. Nodes are dropped based on the ranked projection of features on a learnable vector, $\vec{p}$, as

$$\hat{y}_i = \frac{\mathrm{X}_i \cdot \vec{p}}{\|\vec{p}\|} \qquad \vec{i} = \text{top-k}(\vec{y}, k)$$
$$\mathbf{X}' = \mathrm{X}_{\vec{i}} \odot \tanh(\vec{y_i}) \qquad \mathbf{A}' = \mathbf{A}_{\vec{i};\vec{i}}$$

where $\hat{y}$ are the scores for each node (rows in $\mathbf{X}$) and $\vec{i}$ are the indices of the top-k nodes based on their scores.

Jumping Knowledge Networks In node aggregating schemes, the range of nodes[2] that a node's representation draws from is strongly dependent on the neighbourhood structure (Xu et al., 2018). JK-structures were introduced to allow some flexibility over the degree of aggregation and thus even out the "range" by introducing layer skipping connections. For a node, v, this takes the form

$$h_v^1 = f_1(X_v) \quad ; \quad h_v^i = f_i(h_v^{i-1})$$
$$h_v^{JK} = \text{Agg.}(h_v^1, \ldots, h_v^L)$$

where f_i is the readout of the i^{th} layer and the aggregation function is typically concatenation, summation or an elementwise max, the result being passed to a classifier.

3. Removing JK & Initialisation

Whilst JK-connections were introduced to tackle the problem of node-specific range, in deeper networks they are acting as bypasses of later layers and a hierarchy of representations is not actually being produced. Clearly it runs counter to the core concept of allowing the range to vary over nodes if the higher ranges are not used. To test this we expose the gradient flow and activations in a net of four blocks of GCN+top-k with the final representation aggregated with a global mean and entered into an MLP. ReLU activations are used in the GCN. The GCN weights are initialised using Kaiming (He et al., 2015), while the pools are initialized using Glorot (Glorot & Bengio, 2010)[3]. We refer to this combination as the 'standard initialisation'. Under standard initialisation, layer activations decay into the network, gradients are vanishingly small and the latter part of the network is effectively static under backpropagation.

3.1. ReInit

To expose this problem we propose a data-driven approach similar to LSUV-initialisation (Mishkin & Matas, 2015) to increase activation into the network by maintaining the mean, $\mu(\mathbf{X})$, and variance, $\sigma(\mathbf{X})$, over layers. The idea is simply to initialise under some common scheme and then pass the entire batch through each block, shifting and scaling the layer weights by μ and $k\sigma^{-1}$, respectively, to first zero[4] and then rescale the variance to k, a process we refer to as ReInit. The outputs are related then to the standard-scores (z-values) by the multiplicative term k

$$z(x) = \frac{x - \mu_x}{\sigma_x}$$

$$\mathbf{X}' = k \times z\big(\text{GCN}(\mathbf{X}, \mathbf{A})\big) \quad ; \quad \mathbf{X}'' = k \times z\big(\mathrm{X}'_{\vec{i}} \odot \tanh(\vec{y}_i)\big)$$

this is implemented as affine transforms that are set progressively

$$\mathbf{X}' = \frac{k}{c_1}\big(\text{GCN}(\mathbf{X}, \mathbf{A}) - m_1\big)$$

$$c_1 = \sigma\big(\text{GCN}(\mathbf{X}, \mathbf{A})\big) \quad ; \quad m_1 = \mu\big(\text{GCN}(\mathbf{X}, \mathbf{A})\big)$$

$$\mathbf{X}'' = \frac{k}{c_2}\big(\mathrm{X}'_{\vec{i}} \odot \tanh(\vec{y}_i) - m_2\big)$$

$$c_2 = \sigma\big(\mathrm{X}'_{\vec{i}} \odot \tanh(\vec{y}_i)\big) \quad ; \quad m_2 = \mu\big(\mathrm{X}'_{\vec{i}} \odot \tanh(\vec{y}_i)\big)$$

where the GCN includes the activation function of this layer, with the result that $\sigma(\mathbf{X}') = \sigma(\mathbf{X}'') = k$ and $\mu(\mathbf{X}') = \mu(\mathbf{X}'') = 0$. We deviate from LSUV in not orthonormalising as there is not an analogue that could be applied to the top-k layers so a simple affine transform has a more consistent meaning over the network. The additive constant, m, is introduced to prevent saturation in the $\tanh$ activation. We also found that attempting to derive a semi-analytic solution, in the footsteps of (Glorot & Bengio, 2010), is not possible for the GCN due to the structural asymmetries in neighbourhood aggregation. Essentially, symmetries that allow for the consistent and predictable propagation of variance into-the-depth and over-the-width of dense networks and CNNs are broken in GCNs. The expected variance is sensitive to the number and similarity of neighbours such that properly accounting for these variations requires node-specific information. A data-driven, responsive approach also allows ReInit to be applied on top of any preliminary initialisation scheme, so the 'shape' is not fixed in that sense[5].

[2] Analogous to the receptive field in CNNs.

[3] The authors note the mixed naming conventions here but this seems to be what the community has settled on.

[4] Fix the mean at 0

[5] Although thanks to the repeated aggregation of nodes, so-called 'washing-out', the central-limit theorem can generally be applied.

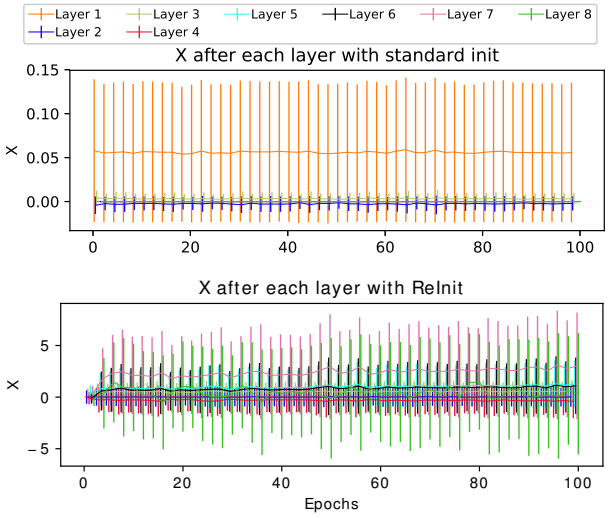

Figure 1. Outputs of each layer during training with the standard initialization (top) and ours (bottom). Note the scale difference. The standard initialization quickly converges to zero for all layers, while with REINIT the values vary widely

4. Shallower, Simpler Networks

To see how much later GCN layers contribute to performance we tested three shallower networks on standard benchmarks. The models could be thought of as extreme ablations.

Structure-blind MLP A three-layer MLP. The adjacency matrix is discarded, the features are globally pooled and passed as input. Three weight matrices, biases; RELU activations. This model cannot see even the number of nodes let alone their individual features or structural relationships.

Single-layer JK GCN+MLP A single layer GCN with a JK-skip preceding the MLP described above. We test this set up both with the weights of the GCN fixed at the random initialization values, denoted (R), and free to update. The fixed method is intended to provide a minimal structural addition to the plain MLP.

5. Experiments & Results

No JK We first present the comparison of activations, gradient flow and training dynamics for a 4-block GNN (as described in 3) in figures 1, 2 & 3, respectively. Detailed analysis of these plots is presented as captions, though the overall picture is that under REINIT training is able to occur whilst under standard initialisation it is not.

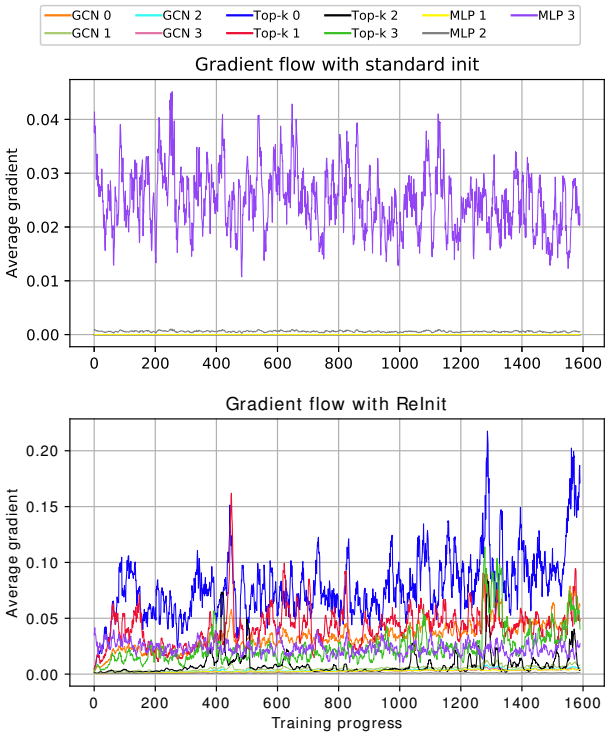

Figure 2. Gradients flowing into the weights of all layers with regular initialization (top) and REINIT (bottom). The gradients of all the layers apart from the last MLP layer are almost 0 for the regular initialization. The reinitialized network manages to train the other layers, although noticeably less gradients flow into the latter layers, possibly by choice rather than a network problem.

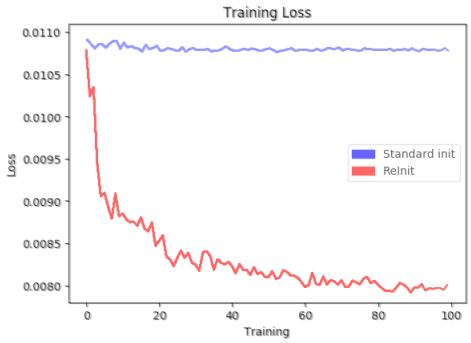

Figure 3. Training loss for the standard initialization and REINIT. The loss does not change for the standard initialization while with REINIT the network is successfully trained.

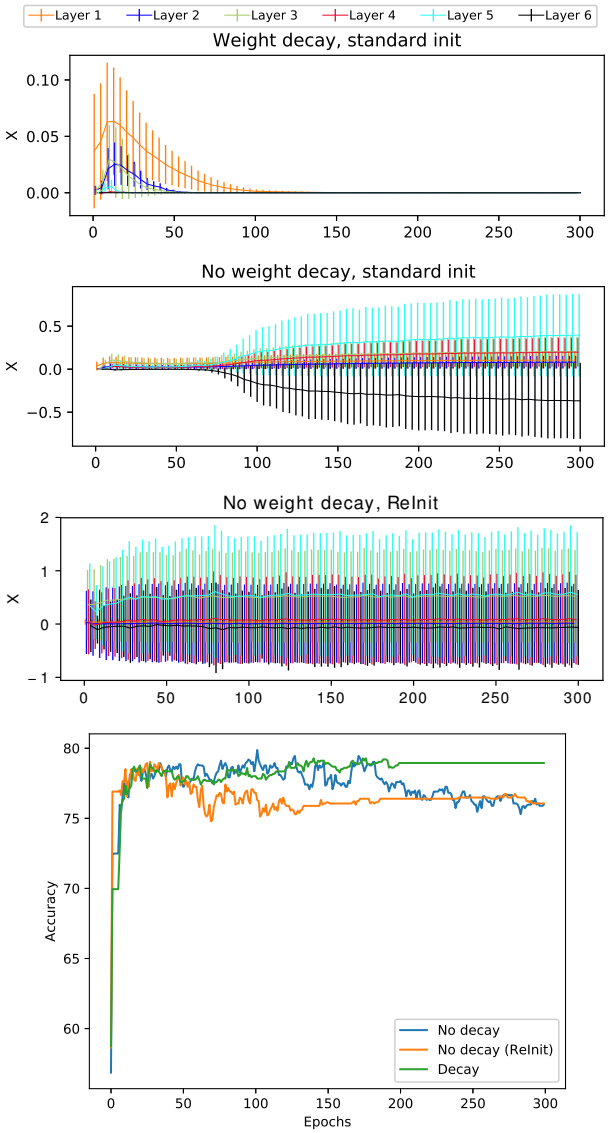

Figure 4. Output values in different training and initialisation routines when training for 300 epochs on the DD dataset. The first plot shows pre-activations vanish in a simple JK-net under standard initialization, trained with Adam with weight decay. The second shows the same network trained without weight decay. The third has no weight decay and is initialized with REINIT. The last figure shows the performance of the three setups on the DD dataset (over 10 folds) as we vary the number of epochs.

5.1. Shallow baselines

We conduct several experiments with the networks described in section 4: a simple MLP; a randomly initialized GCN, which is not updated during the training process, denoted GCN(R)-MLP; and a GCN that is free to update (GCN-MLP).

We find that these models surpass most of the previous methods. In some cases surpassing even the recent differentiable pooling methods. We note that the performance of the random GCN should not come as a surprise given its connection to WL-test (Kipf & Welling, 2016). This is most relevant in the case of the random GCN, having very little power in the featural domain but adding structural information comparable to 1-WL.

Table 1. Classification accuracy percentages. The results of other networks are taken from Cangea et al. 2018 with which we share 10-fold splits for benchmarking our methods. Bold indicates top-performance, blue indicates weaker performance than the MLP.

	DATASETS			
MODEL	REDDIT [7]	DD	COLLAB	PROT.
PATCHYSAN	41.32	76.27	72.60	75.00
GRAPHSAGE	42.24	75.42	68.25	70.48
ECC	41.73	74.10	67.79	72.65
SET2SET	43.49	78.12	71.75	74.29
SORTPOOL	41.82	79.37	73.76	75.54
DIFFPOOL-DET	46.18	75.47	**82.13**	75.62
DIFFPOOL-NOLP	46.65	79.98	75.63	77.42
DIFFPOOL	47.08	**81.15**	75.50	**78.10**
GU-NET/SHGC	-	78.59	74.54	75.46
MLP	40.96	80.22	74.00	75.74
GCN(R)-MLP	36.15	78.61	75.38	76.28
GCN-MLP	45.01	79.29	76.50	75.64
JK-SUM	**47.16**	79.02	77.00	75.82
JK-SUM-DECAY	43.87	79.11	74.14	75.82
JK-SUM-REINIT	46.77	75.97	77.20	75.46

These initial results (presented in table 1) show that there is room for advancements in graph classification and that these simple models are to be considered strong baselines. These networks, particularly the MLP, are simple and appear as subnetworks in many methods. As such, it is of paramount importance to undertake thorough ablation studies to show the benefit of complexifying networks. For instance, we can add additional components that improve upon other approaches but do so by relying heavily on these simpler subnetworks. We explore this idea below.

5.2. Excessively deep networks

We use the following architecture in the next few experiments: GCN-POOL-GCN-POOL-GCN-POOL-MLP with the global max and sum of each layer passed to the MLP through JK-structures. Due to the initialization problem, if weight decay is used[6] the network is unable to recover from a bad initialization and as such it cannot learn in the deeper layers (see Figure 4). This method (JK-SUM-DECAY) is competitive with most results, performing closely to the simple sub-network it contains: GCN-MLP.

[6]Here we use $\lambda = 5{\times}10^{-3}$ with a learning rate of $5{\times}10^{-4}$ but smaller values achieve similar results.

Next, even if we do not use any weight decay the network will only be able to recover the deeper layers after a significant number of epochs. For instance, for DD the network only starts to recover the deeper layers after epoch 100 as shown in Figure 4. Although, to fully recover the layers (similarly to the network with REINIT) we found that the network needs to be trained for more than 800 epochs and, if early-stopping causes training to end in an earlier epoch, we would still be using only the first two layers (GCN+POOL). In fact, the optimal number of epochs to train the network for was 100 which is what we report in the results in Table 1 (JK-SUM). However, the network behaves very differently when initialized using REINIT as the method does not need to recover the layers one-by-one, changing the dynamics and ultimately how and what the network learns. The same figure shows that in the case of REINIT all the layers are trainable from the beginning. In that case, we notice that the performance goes up sharply in the very first few epochs for DD (less than 10, see last plot of Figure 4) and then drops and converges to roughly the same as the recovered network with standard initialization (without weight decay). While for small datasets (DD, PROTEINS) unleashing the power of the deeper network from the beginning is not beneficial since it can cause over-fitting (a single layer GCN already performs well) for COLLAB we see that this differs. In fact, for these small datasets, the method with REINIT achieved highest accuracy in fewer than 50 epochs, while for COLLAB it was 300. The same network without REINIT had the best performance training for 100 epochs, but resulted in a lower quality model. This hints that for this bigger dataset all 3-layers are needed, while for smaller problems the network is likely over-parameterised and this is exposed by REINIT.

Closing remarks We have demonstrated that some very simple models are competitive with the SOTA and that JK-structures may permit models to perform well through these subnetworks. We hope that these baselines and a greater interest in ablation studies will be adopted by the community.

References

Bronstein, M. M., LeCun, Y., Szlam, A., Vandergheynst, P., and Bruna, J. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 11 2017. ISSN 1053-5888. doi: 10.1109/msp.2017.2693418. URL http://arxiv.org/abs/1611.08097http://dx.doi.org/10.1109/MSP.2017.2693418.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. Towards Sparse Hierarchical Graph Classifiers. 11 2018. URL http://arxiv.org/abs/1811.01287.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gao, H. and Ji, S. Graph u-net, 2019. URL https://openreview.net/forum?id=HJePRoAct7.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks, 3 2010. ISSN 1938-7228. URL http://proceedings.mlr.press/v9/glorot10a.html.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudk, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2 2015. URL http://arxiv.org/abs/1502.01852.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.90. URL http://dx.doi.org/10.1109/CVPR.2016.90.

Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. 9 2016. URL http://arxiv.org/abs/1609.02907.

Luzhnica, E., Day, B., and Lio', P. Clique pooling for graph classification. 3 2019. URL http://arxiv.org/abs/1904.00374.

Mishkin, D. and Matas, J. All you need is a good init. 11 2015. URL http://arxiv.org/abs/1511.06422.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs, 2016.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.11. URL http://dx.doi.org/10.1109/CVPR.2017.11.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation Learning on Graphs with Jumping Knowledge Networks. 6 2018. URL http://arxiv.org/abs/1806.03536.

Ying, R., You, J., Morris, C., Ren, X., Hamilton, W. L., and Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. 2018. ISSN 18160948. doi: arXiv:1806.08804v3. URL https://arxiv.org/pdf/1806.08804.pdfhttp://arxiv.org/abs/1806.08804.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.